NETWORK PROGRAMMING III - SOCKETSERVER

Note

This chapter is based on socketserver - A framework for network servers.

Echo Server

In this section, we'll create an echo server using **socketserver** which is a module that simplifies the task of writing network servers. Actually, the **socketserver** is a framework for network servers. For a version < 3.x.x, **SocketServer** should be used instead of **socketserver**.

The server we're creating echoes the message received from clients except it sends the message back upper-cased.

Echo Server code

```
# echo_server.py

import SocketServer

class MyTCPSocketHandler(SocketServer.BaseRequestHandler):
    """
    The RequestHandler class for our server.
    It is instantiated once per connection to the server, and must
    override the handle() method to implement communication to the
    client.
    """

    def handle(self):
        # self.request is the TCP socket connected to the client
        self.data = self.request.recv(1024).strip()
        print("{} wrote:".format(self.client_address[0]))

        print(self.data)
        # just send back the same data, but upper-cased
        self.request.sendall(self.data.upper())

if __name__ == "__main__":

    HOST, PORT = "localhost", 9999

    # instantiate the server, and bind to localhost on port 9999
    server = SocketServer.TCPServer((HOST, PORT), MyTCPSocketHandler)

    # activate the server
    # this will keep running until Ctrl-C
    server.serve_forever()
```

Here are the steps to take to create a server:

1. We must create a request handler class by subclassing the **BaseRequestHandler**class.

```
class MyTCPSocketHandler(SocketServer.BaseRequestHandler):
```

2. The child class should override the inherited **handle()** method.

```
def handle(self):
    # self.request is the TCP socket connected to the client
    self.data = self.request.recv(1024).strip()
    print("{} wrote:".format(self.client_address[0]))
    print(self.data)
    # just send back the same data, but upper-cased
    self.request.sendall(self.data.upper())
```

3. The **handle()** method will process incoming requests.
4. We must instantiate one of the server classes, passing it the server's address and the request handler class.

```
server = SocketServer.TCPServer((HOST, PORT), MyTCPSocketHandler)
```

5. Finally, call the **handle_request()** or **serve_forever()** method of the server object to process one or many requests.

Echo client code

```
# echo_client.py

import socket, sys

HOST, PORT = "localhost", 9999
data = " ".join(sys.argv[1:])
print 'data = %s' %data

# create a TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # connect to server
    sock.connect((HOST, PORT))

    # send data
    sock.sendall(bytes(data + "\n"))

    # receive data back from the server
    received = str(sock.recv(1024))
finally:
    # shut down
    sock.close()

print("Sent:     {}".format(data))
print("Received: {}".format(received))
```

Running the Echo server / client code

Here are the outputs:

Server:

```
$ python s.py
127.0.0.1 wrote:
Hello from a client
```

Client:

```
$ python c.py Hello from a client
data = Hello from a client
Sent:     Hello from a client
Received: HELLO FROM A CLIENT
```

Python Network Programming

Network Programming - Server & Client A : Basics

Network Programming - Server & Client B : File Transfer

Network Programming II - Chat Server & Client

Network Programming III - SocketServer

Network Programming IV - SocketServer Asynchronous request